AMENDMENTS TO THE CLAIMS:

This listing of claims will replace all prior versions, and listings, of claims in the application:

1. (Canceled)

2. (Previously Presented) A method according to Claim 21, wherein the predefined number is variable from one predefined block of instructions to another.

3. (Previously Presented) A method according to claim 21, wherein the common set of instructions comprises at least one calculation instruction that is equivalent to a calculation instruction of each predefined block in the context of a covert channel attack.

4. (Previously Presented) A method according to Claim 3, in which the common set of instructions also comprises an instruction to update a loop pointer indicating a number of executions already performed with the common set of instructions.

5. (Previously Presented) A method according to Claim 3 wherein the common set of instructions also comprises an instruction to update a state pointer indicating whether the predefined number has been reached.

6. (Previously Presented) A method according to Claim 4 wherein the value of the loop pointer is a function of the value of the input variable and/or of the number of instructions in the selected block of instructions.

7. (Previously Presented) A method according to claim 21, wherein, in order to successively effect several blocks of instructions chosen from amongst the plural predefined blocks of instructions, each selected block of instructions is selected as a function of an input variable associated with an input index, and

the common set of instructions is executed a total number of times equal to a sum of the predefined numbers associated with each selected block of instructions.

8. (Previously Presented) A method according to Claim 7 wherein one and the same block of instructions is selected several times according to the input variable associated with the input index.

9. (Previously Presented) A method according to claim 7, wherein at least two of the following data items, (a) the value of a loop pointer, (b) the value of a state pointer, (c) the value of the input variable, and (d) the number of instructions of the selected block of instructions, are linked by one or more mathematical functions.

10. (Previously Presented) A method according to Claim 9, used in the implementation of an exponentiation calculation of the type $B = A^D$, with D being an integer number of M bits, and each bit (D_i) of D corresponding to an input variable of input index i, comprising the following steps:

Initialisation:

```
    $R_0$ <- 1; $R_1$ <- A; i <- M-1
As long as $i \geq 0$, repeat the common set of instructions:
    k <- (/s)x(k+1) + sx2x(/$D_i$)
    s <- (k mod 2) + (k div 2)
    $\gamma(k,s)$:    $R_0$ <- $R_0$x$R_k$ mod 2
                      i <- i – s
Return $R_0$,
```

where R_0 and R_1 are values stored in two registers, respectively,

k is a loop pointer indicating a number of executions performed with the common set of instructions, and

s is a state pointer indicating whether the predefined number has been reached.

11. (Previously Presented) A method according to Claim 9, used in the implementation of an exponentiation calculation of the type $B = A^D$, with D being an integer number of M bits, and each bit (D_i) of D corresponding to an input variable of input index i, comprising the following steps:

Initialisation:

```
    $R_0$ <- 1; $R_1$ <- A; i <- M-1; k <- 1
As long as $i \geq 0$, repeat the common set of instructions:
    k <- (Di) AND (/k)
    $\gamma'(s,k)$:    $R_0$ <- $R_0$x$R_k$
                       i <- i – (/k)
```

Return R_0,

where R_0 and R_1 are values stored in two registers, respectively,

k is a loop pointer indicating a number of executions performed with the common set of instructions, and

s is a state pointer indicating whether the predefined number has been reached.

12. (Previously Presented) A method according to Claim 9, used in the implementation of an exponentiation calculation of the type $B = A^D$, with D being an integer number of M bits, and each bit (D_i) of D corresponding to an input variable of input index i, comprising the following steps:

```
Initialisation:
    $R_0$ <- 1; $R_1$ <- A; i <- 0; k <- 1
As long as $i \leq M-1$, repeat the common set of instructions:
    k <- $k \oplus D_i$
    $\gamma(k)$:  $R_k$ <- $R_k x R_1$
             i <- i+k
Return $R_0$,
```

where R_0 and R_1 are values stored in two registers, respectively, and

k is a loop pointer indicating a number of executions performed with the common set of instructions.

13. (Previously Presented) A method according to Claim 9, used in the implementation of an exponentiation calculation of the type $B = A^D$, with D being an

integer number of M bits, and each bit (D_i) of D corresponding to an input variable of input index i, comprising the following steps:

```
Initialisation:
    $R_0 <- 1; R_1 <- A,; R_2 <- A^3;$
    $D_{-1} <- 0; i <- M-1; s <- 1$
As long as $i \geq 0$, repeat the common set of instructions:
    $k <- (/s)x(k+1) + sx(D_i + 2x(D_i \text{ AND } D_{i-1}))$
    $s <- /((k \bmod 2) \oplus (k \text{ div } 4))$
    $\gamma(k,s)$:      $R_0 <- R_0xR_{sx(k \text{ div } 2)}$
                  $i <- i - sx(k \bmod 2 + 1)$
Return $R_0$,
```

where R_0 and R_1 are values stored in two registers, respectively,

k is a loop pointer indicating a number of executions performed with the common set of instructions, and

s is a state pointer indicating whether the predefined number has been reached.

14. (Previously Presented) A method according to Claim 9, used in the implementation of an exponentiation calculation of the type $B = A^D$, with D being an integer number of M bits, and each bit (D_i) of D corresponding to an input variable of input index i, comprising the following steps:

```
Initialisation:
    $R_0 <- 1; R_1 <- A; R_2 <- A^3;$
    $D_{-1} <- 0; i <- M-1; s <- 1$
```

```
As long as $i \geq 0$, repeat:
        $k \leftarrow (/s)x(k+1)$
        $s \leftarrow s \oplus D_i \oplus ((D_{i-1}$ AND $(k \bmod 2))$
                $R_0 \leftarrow R_0 x R_{kxs}$
                $i \leftarrow i - kxs - (/D_i)$
Return $R_0$,
```

where R_0 and R_1 are values stored in two registers, respectively,

k is a loop pointer indicating a number of executions performed with the common set of instructions, and

s is a state pointer indicating whether the predefined number has been reached.

15. (Previously Presented) A method according to claim 7, wherein at least two of the following data items, (a) the value of a loop pointer, (b) the value of a state pointer, (c) the value of the input variable, and (d) the number of instructions of the selected block of instructions, are linked and such linking is defined by a table with several inputs.

16. (Previously Presented) A method according to Claim 15, used in the implementation of an exponentiation calculation of the type $B = A^D$, with D being an integer number of M bits, and each bit (D_i) of D corresponding to an input variable of input index i, comprising the following step:

```
As long as $i \geq 0$, repeat the common set of instructions:
        $k \leftarrow (/s)x(k+1) + sx2x(/D_i)$
```

```
s <- U(k,1)
γ(k,s):        $R_0$ <- $R_0 x R_{U(k, 0)}$
               i <- i – s
```

where (U(k,1)) is the following matrix:

$$(U(k,1))_{\substack{0 \le k \le 2 \\ 0 \le l \le 1}} = \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 1 \end{pmatrix},$$

R_0 and R_1 are values stored in two registers, respectively,

k is a loop pointer indicating a number of executions performed with the common set of instructions, and

s is a state pointer indicating whether the predefined number has been reached.

17. (Previously Presented) A method according to Claim 15, used in the implementation of an exponentiation calculation of the type $B = A^D$ according to the algorithm (M, M^3), with D being an integer number of M bits, and each bit (D_i) of D corresponding to an input variable of input index i, comprising the following step:

As long as $i \geq 0$, repeat the common set of instructions:

```
k <- (/s)x(k+1) + sx($D_i$ + 2x(/$D_i$ AND $D_{i-1}$))
s <- U(k,2)
γ(k,s):        $R_0$ <- $R_0 x R_{U(k,0)}$;
               i <- i – U(k,1)
```

where (U(k,1)) is the following matrix:

$$(U(k,1))_{\substack{0 \le k \le 5 \\ 0 \le 1 \le 2}} = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 2 & 1 \end{pmatrix},$$

R_0 and R_1 are values stored in two registers, respectively,

k is a loop pointer indicating a number of executions performed with the common set of instructions, and

s is a state pointer indicating whether the predefined number has been reached.

18. (Previously Presented) A method according to Claim 15, used in the implementation of a calculation on an elliptic curve in affine coordinates, a calculation using operations of the addition or doubling of points type, and in which the following step is performed:

As long as $i \ge 0$, repeat:

```
$\gamma(k)$:  $R_{U(k,0)} \leftarrow R_1 + R_3$;
        $R_{U(k,1)} \leftarrow R_{U(k,1)} + R_{U(k,2)}$;
        $R_5 \leftarrow R_2/R_1$; $R_{U(k,3)} \leftarrow R_1 + R_5$;
        $R_{U(k,4)} \leftarrow R_5^2$;
        $R_{U(k,4)} \leftarrow R_{U(k,4)} + a$;
        $R_1 \leftarrow R_1 + R_{U(k,5)}$;
        $R_2 \leftarrow R_1 + R_{U(k,6)}$; $R_6 \leftarrow R_1 + R_{U(k,7)}$;
        $R_5 \leftarrow R_5 . R_6$; $R_2 \leftarrow R_2 + R_5$
$s \leftarrow k - D_i + 1$
```

k <- (k+1) x (/s);

i <- i – s;

where (U(k,1)) is the following matrix:

(U(k,1)) (U(k,1))

$$(U(k,1))_{\substack{0 \le k \le 1 \\ 0 \le 1 \le 10}} = \begin{pmatrix} 1 & 2 & 4 & 1 & 6 & 6 & 4 & 3 \\ 6 & 6 & 3 & 5 & 1 & 5 & 2 & 6 \end{pmatrix},$$

R_0 and R_1 are values stored in two registers, respectively,

k is a loop pointer indicating a number of executions performed with the common set of instructions, and

s is a state pointer indicating whether the predefined number has been reached.

19. (Previously Presented) A method for obtaining an elementary set of instructions common to a plurality of predefined blocks of instructions, for implementing a cryptographic calculation method according to claim 21, comprising the following steps:

E1: breaking down each predefined block of instructions into a series of elementary blocks that are equivalent in the context of a covert channel attack, and classifying all the elementary blocks,

E2: identifying a common elementary block that is equivalent to all the elementary blocks of all the predefined blocks of instructions,

E3: identifying a common block comprising at least the common elementary block previously identified and an instruction to update a loop pointer such that an execution of the common elementary block associated with the value of the loop pointer and an execution of the elementary block with a rank equal to the value of the loop pointer are identical.

20. (Previously Presented) A method according to Claim 19, wherein, during step E1, at least one fictional instruction is added to at least one predefined block of instructions.

21. (Currently Amended) A method for implementing a cryptographic calculation in an electronic device, comprising the following steps:

selecting a block of instructions from amongst a plurality of predefined blocks of instructions, as a function of an input variable; and

executing, in the electronic device, a set of instructions that is common to the plurality of predefined blocks of instructions a predefined number of times, wherein said predefined number is associated with the selected block of instructions.

22. (Previously Presented) A method according to claim 5, wherein the value of the state pointer is a function of the value of the input variable and/or of the number of instructions in the selected block of instructions.

23. (Previously Presented) A method according to claim 15, wherein said several inputs comprise a matrix.

24. (Previously Presented) A method according to claim 21, wherein said electronic device is a chip card.